学籍番号202113517

所属 情報学群情報メディア創成学類 3年

氏名 高橋 健太郎

(1)

#include <cstdlib>

#include <cmath>

#include <vector>

#include <GLUT/glut.h>

*// 2次元ベクトルを扱うためのクラス*

**class** Vector2d

{

**public:**

**double** x, y;

Vector2d() { x = y = 0; }

Vector2d(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

**void** set(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

*// 長さを1に正規化する*

**void** normalize()

{

**double** len = length();

x /= len;

y /= len;

}

*// 長さを返す*

**double** length() { return sqrt(x \* x + y \* y); }

*// s倍する*

**void** scale(**const** **double** s)

{

x \*= s;

y \*= s;

}

*// 加算の定義*

Vector2d operator+(Vector2d v) { return Vector2d(x + v.x, y + v.y); }

*// 減算の定義*

Vector2d operator-(Vector2d v) { return Vector2d(x - v.x, y - v.y); }

*// 内積の定義*

**double** operator\*(Vector2d v) { return x \* v.x + y \* v.y; }

*// 代入演算の定義*

Vector2d **&**operator=(**const** Vector2d **&**v)

{

x = v.x;

y = v.y;

return (\*this);

}

*// 加算代入の定義*

Vector2d **&**operator+=(**const** Vector2d **&**v)

{

x += v.x;

y += v.y;

return (\*this);

}

*// 減算代入の定義*

Vector2d **&**operator-=(**const** Vector2d **&**v)

{

x -= v.x;

y -= v.y;

return (\*this);

}

*// 値を出力する*

**void** print() { printf("Vector2d(%f %f)\n", x, y); }

};

*// マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる*

Vector2d operator-(**const** Vector2d **&**v) { return (Vector2d(-v.x, -v.y)); }

*// ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる*

Vector2d operator\*(**const** **double** **&**k, **const** Vector2d **&**v) { return (Vector2d(k \* v.x, k \* v.y)); }

Vector2d operator\*(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x \* k, v.y \* k)); }

*// ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる*

Vector2d operator/(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x / k, v.y / k)); }

*// ================================================================================================*

std::vector<Vector2d> g\_ControlPoints; *// 制御点を格納する*

*// ノットベクトルの要素数 （参考書にあわせて、要素数は10としている）*

**const** **int** NUM\_NOT = 10;

*// ノットベクトル*

*// この配列の値を変更することで基底関数が変化する。その結果として形が変わる。*

*// 下の例では、一定間隔で値が変化するので、「一様Bスプライン曲線」となる*

**double** g\_NotVector[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

*// 基底関数 N{i,n}(t)の値を計算する*

**double** getBaseN(**int** i, **int** n, **double** t)

{

if (n == 0)

{

*// n が 0 の時だけ t の値に応じて 0 または 1 を返す*

if (t >= g\_NotVector[i] && t < g\_NotVector[i + 1])

{

return 1.0;

}

return 0;

}

else

{

*// ★ここに必要なプログラムコードを記述する*

*// ★再帰（自分自身の関数 getBaseN を呼ぶ処理が必要）*

*// ★係数を計算するときに、ノットが重なる（分母がゼロとなる）ときには、その項を無視する。*

double temp1, temp2;

if ((g\_NotVector[i + n] - g\_NotVector[i]) != 0)

{

temp1 = ((t - g\_NotVector[i]) / (g\_NotVector[i + n] - g\_NotVector[i]));

}

else

{

temp1 = 0;

}

if ((g\_NotVector[i + n + 1] - g\_NotVector[i + 1]) != 0)

{

temp2 = ((g\_NotVector[i + n + 1] - t) / (g\_NotVector[i + n + 1] - g\_NotVector[i + 1]));

}

else

{

temp2 = 0;

}

return temp1 \* getBaseN(i, n - 1, t) + temp2 \* getBaseN(i + 1, n - 1, t);

}

}

*// 表示部分をこの関数で記入*

**void** display(**void**)

{

glClearColor(1.0, 1.0, 1.0, 1.0); *// 消去色指定*

glClear(GL\_COLOR\_BUFFER\_BIT); *// 画面消去*

*// 制御点の描画*

glPointSize(5);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// 制御点を結ぶ線分の描画*

glColor3d(1.0, 0.0, 0.0);

glLineWidth(1);

glBegin(GL\_LINE\_STRIP);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// ★ ここにBスプライン曲線を描画するプログラムコードを入れる*

*// ヒント1: 3次Bスプラインの場合は制御点を4つ入れるまでは何も描けない*

*// ヒント2: パラメータtの値の取り得る範囲に注意*

glColor3d(1.0, 0.0, 0.0);

glLineWidth(1);

glBegin(GL\_LINE\_STRIP);

for (unsigned int t = 4; t <= g\_ControlPoints.size(); t += 1)

{

double px = 0, py = 0;

for (unsigned int i = 0; i <= 3 + g\_ControlPoints.size() - 1; i++)

{

px += getBaseN(i, 3, t) \* g\_ControlPoints[i].x;

py += getBaseN(i, 3, t) \* g\_ControlPoints[i].y;

}

glVertex2d(px, py);

}

glEnd();

glutSwapBuffers();

}

**void** resizeWindow(**int** w, **int** h)

{

h = (h == 0) ? 1 : h;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

*// ウィンドウ内の座標系設定*

*// マウスクリックの座標と描画座標が一致するような正投影*

glOrtho(0, w, h, 0, -10, 10);

glMatrixMode(GL\_MODELVIEW);

}

*// キーボードイベント処理*

**void** keyboard(**unsigned** **char** key, **int** x, **int** y)

{

switch (key)

{

case 'q':

case 'Q':

case '\033':

exit(0); */\* '\033' は ESC の ASCII コード \*/*

default:

break;

}

glutPostRedisplay();

}

*// マウスイベント処理*

**void** mouse(**int** button, **int** state, **int** x, **int** y)

{

if (state == GLUT\_DOWN)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:

*// クリックした位置に制御点を追加*

*// ノット数を増やせばいくらでも制御点を追加できるが、今回はNUM\_NOTの値で固定されているので*

*// いくらでも追加できるわけではない*

if (g\_ControlPoints.size() < NUM\_NOT - 4)

{

g\_ControlPoints.push\_back(Vector2d(x, y));

}

break;

case GLUT\_MIDDLE\_BUTTON:

break;

case GLUT\_RIGHT\_BUTTON:

*// 末尾の制御点の削除*

if (!g\_ControlPoints.empty())

{

g\_ControlPoints.pop\_back();

}

break;

default:

break;

}

glutPostRedisplay(); *// 再描画*

}

}

*// メインプログラム*

**int** main(**int** argc, **char** **\***argv[])

{

glutInit(&argc, argv); *// ライブラリの初期化*

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE); *// 描画モードの指定*

glutInitWindowSize(800, 800); *// ウィンドウサイズを指定*

glutCreateWindow(argv[0]); *// ウィンドウを作成*

glutDisplayFunc(display); *// 表示関数を指定*

glutReshapeFunc(resizeWindow); *// ウィンドウサイズが変更されたときの関数を指定*

glutKeyboardFunc(keyboard); *// キーボードイベント処理関数を指定*

glutMouseFunc(mouse); *// マウスイベント処理関数を指定*

glutMainLoop(); *// イベント待ち*

return 0;

}

(3)

ノットの数を増やすと、現れる曲線の数が増える。

ノットの値を変更すると、曲線の向きが変わる。